Machine Learning Road Map

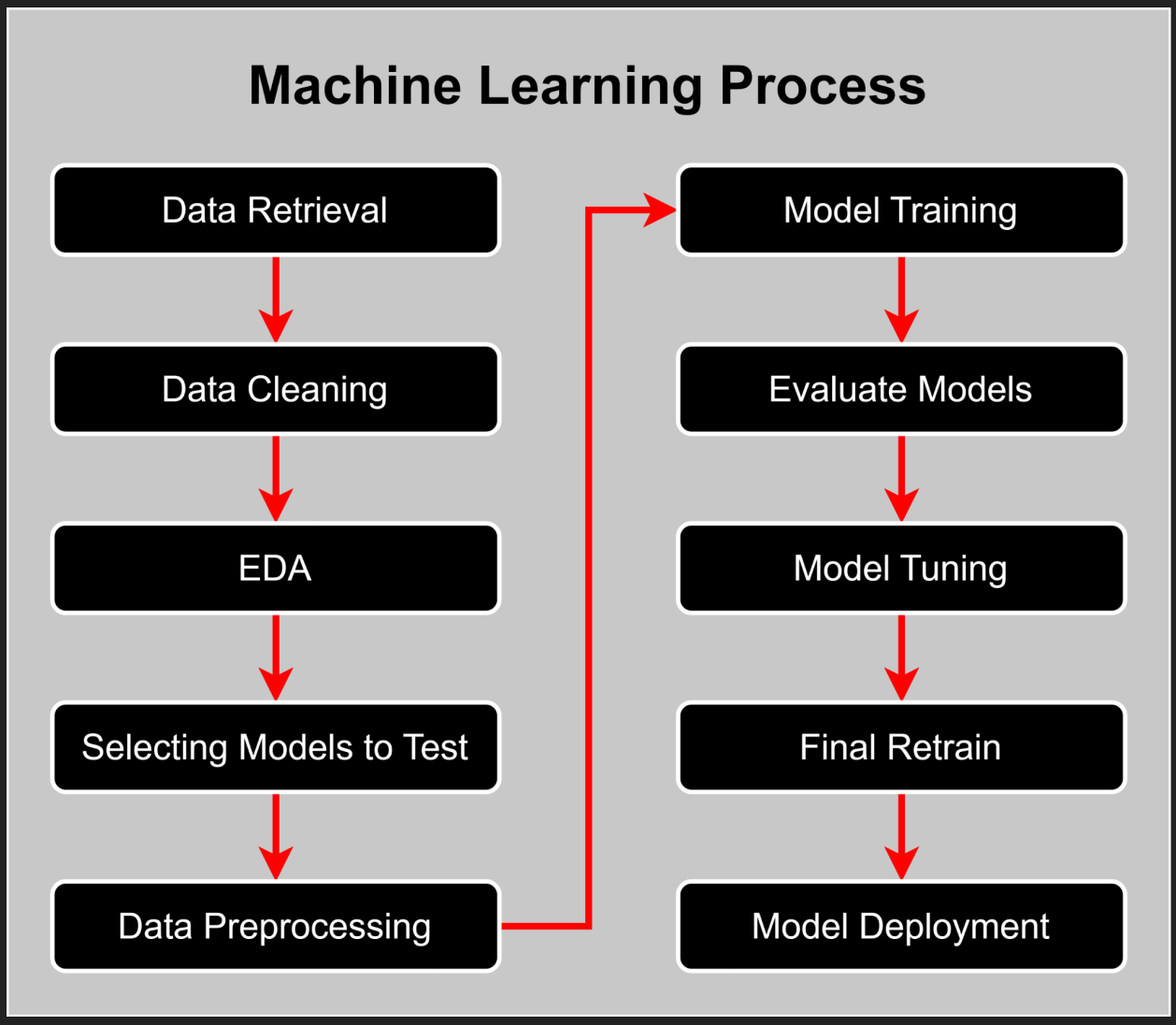
A guide for non-PHDs

Pacheco
Ahmad
Pruitt

Table of Contents

About this Guide

This document is not intended to be a *full guide* to machine learning, but a high-level road map to the overall process. The ideal approach to using this document is to treat each chapter as a step in the machine learning process, with the chapter containing an overview of the many things to consider, sample code, and links to additional resources where appropriate. No one chapter will be particularly detailed, but should provide sufficient information to kickstart further research.



Data Selection Process

The initial step in any machine learning project is obviously to gather the data which will be used to train the model. It is worth considering the corpus of this document,

particularly as it applies to the arrangement and preparation of data, while performing this selection. A tremendous amount of time can be saved simply by seeking data to use from the start that is close to what will be required for the model.

Data Preparation

{{ TEXT }}

Data Cleaning

After selecting a dataset, cleaning it up and preparing it for use is the critical next step. This step is often the longest and most intensive to complete. Some integral elements to keep in mind during this step are:

Removing Bad Data

1. **NaNs**
   How many nulls are in each column of the data? A good rule of thumb is if there are more than 70% null values in a column, drop it. Less than that, look deeper into the data. Is it a seemingly important feature? If not, it may be easiest to simply drop it. If it does seem important, consider imputing the data:

   Useful links:
   - Seven Ways to Make up Data: Common Methods to Imputing Missing Data - The Analysis Factor

2. **Duplicates**
   It is critical to ensure the dataset is free from any duplicate **rows** of data and any records showing duplicates with **conflicting** data should be removed entirely.

3. **Erroneous Data**
   Anywhere you find data that seems erroneous or dubiously trustworthy, it is important to remove it from your dataset.

Exploratory Data Analysis

The Exploratory Data Analysis, or EDA, is where we start to perform some real statistical analysis on our data. The EDA is the process of examining and summarizing the main characteristics of a dataset to gain insights and discover patterns and relationships that can inform subsequent analysis. In the context of machine learning, the goal is to identify useful pieces of information and relationships within our data in

order to prepare our data for modeling such that it can answer our question or solve our problem.

During an EDA, various statistical techniques and visualizations can be used to gain insights into the data. These may include summary statistics such as mean, median, mode, and standard deviation, histograms, scatter plots, box plots, heat maps, and correlation matrices. EDA can also involve data cleaning and testing out various preprocessing steps, such as imputing missing values, scaling and normalizing data, and encoding categorical variables.

What is the Target?

The target variable, also known as the dependent variable, is the variable in a machine learning problem that the algorithm is trying to predict or estimate based on the input variables, which are also known as independent variables or features. In supervised learning, the target variable is labeled and the algorithm learns to map the input variables to the corresponding output or label. Identifying the target variable is critical to begin consideration of what model to use.

Next, consider the kind of data the target represents. Is it Quantitative or Qualitative?

- **Quantitative Data** is strictly data composed of numerical observations.
    - **Discrete Data** is data that can only be certain values, such as a range of integers from 1 to 10.
    - **Continuous Data** is data that can be any real number (as in decimals) across a range.
- **Qualitative Data** is descriptive and written as text that cannot be measured empirically. Examples are “green” vs. “red” or “happy” vs. “sad”.

In contrast, unsupervised learning does not have a target variable, and the algorithm learns to identify patterns and structure in the data without any prior knowledge of the correct output.

Handling Outliers

Remove, scale, or leave alone? Depending on the nature of the dataset and the question it is being used to answer, these outliers may be highly desirable features in the dataset. If so, considerations should be made for how this will impact any need for

scaling of the data. If not, evaluate the outliers individually. These may be the result of more erroneous data.

Relationships In The Data

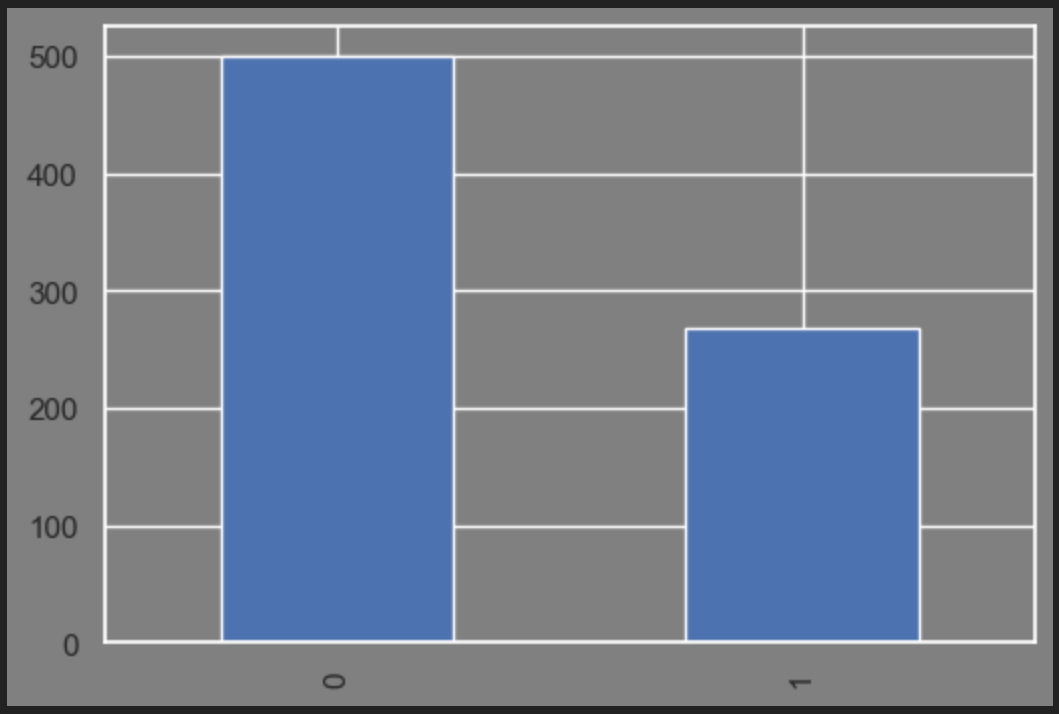
{{ TEXT }}

Balance/Imbalance In Target

Is the dataset balanced? That is, are the target classes' ratios proportional to each other? In machine learning, a balanced dataset is one in which the number of samples for each class in the target feature is roughly the same, while an imbalanced dataset is one in which one or more classes are significantly underrepresented. Most likely, this will not be the case. The importance of having a balanced dataset lies in the fact that machine learning algorithms can be biased towards the majority class in imbalanced datasets, leading to poor performance for the minority class. Moreover, a model trained on an imbalanced dataset may overfit to the majority class and make inaccurate predictions for the minority class. To check if a dataset is balanced, we can calculate the number of samples for each class and compare them.

```
# Importing our Dependencies
import pandas as pd

# Load the dataset into a Pandas DataFrame
df = pd.read_csv('dataset.csv')

# Count the number of samples for each class
class_counts = df['Target'].value_counts()
print(class_counts)

# Display the counts in bar chart
class_counts.plot(kind='bar')
plt.show()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```



Using pandas to count the number of classes in the target feature.

If the output of this code shows that the number of samples for each class is roughly the same, the dataset is balanced. If the number of samples for one class is much smaller than the other, the dataset is imbalanced. It is a good idea to combine this numerical analysis with visualizations to give yourself an intuitive understanding of your data.

If a dataset is imbalanced, there are several techniques that can be used to balance it. One common technique is oversampling, which involves creating additional samples for the minority class. Below is an example of how to use the `imblearn` library to oversample the minority class of a binary classification dataset:

```
# Importing our Dependencies
import pandas as pd
from imblearn.over_sampling import RandomOverSampler

# Load the dataset into a Pandas DataFrame
df = pd.read_csv('dataset.csv')

# Split the dataset into features and target
X = df.drop('target', axis=1)
y = df['target']
```

```
# Oversample the minority class
ros = RandomOverSampler()
X_resampled, y_resampled = ros.fit_resample(X, y)
```

Annotation

This code uses the RandomOverSampler class from the `imblearn` library to create additional samples for the minority class. The fit_resample() method is then called to oversample the dataset. After oversampling, the number of samples for each class should be roughly the same, resulting in a balanced dataset. Other techniques for balancing imbalanced datasets include undersampling, synthetic data generation, and cost-sensitive learning.

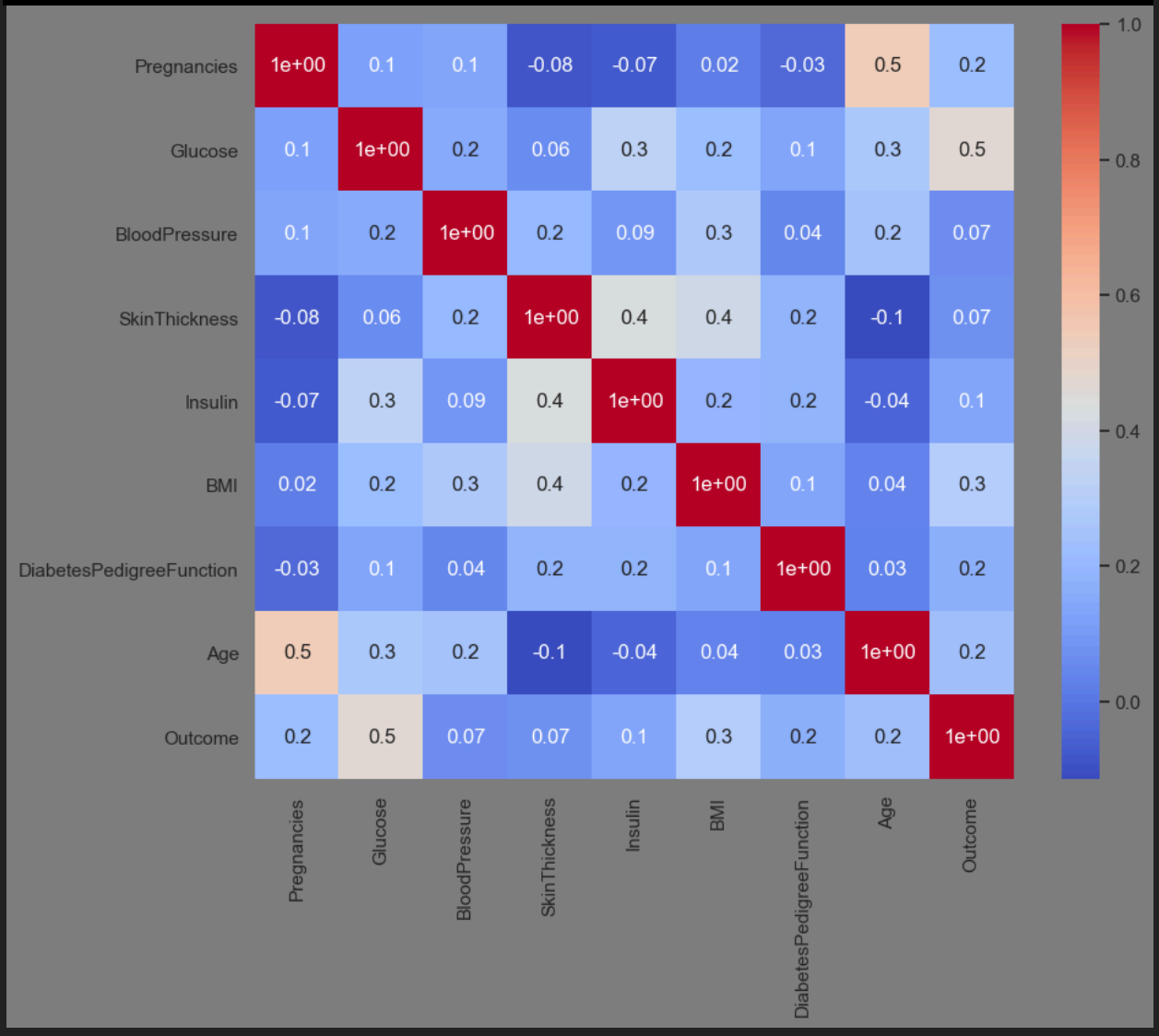
Feature Relevance

Feature relevance in machine learning is the process of determining which features or variables in a dataset are most relevant or influential for predicting the target variable. This is important because including irrelevant or redundant features can decrease the accuracy of the model and increase the risk of overfitting, while excluding important features can result in a less accurate model. Feature relevance can be determined using various statistical methods, such as correlation analysis and feature importance scores.

A common method for initially exploring feature relevance is to create a correlation matrix with the dataset. A correlation matrix is a table that shows the pairwise correlation coefficients between each pair of variables in a dataset. The correlation coefficient is a measure of the strength and direction of the linear relationship between two variables, ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation), with 0 indicating no correlation. A correlation matrix can be used to identify which features are strongly correlated with each other, which can help to avoid multicollinearity, a situation where two or more features are highly correlated, making it difficult for the algorithm to distinguish their individual effects on the target variable.

```
# Importing our Dependencies
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the data into a pandas DataFrame
df = pd.read_csv('diabetes.csv')

# Create a correlation matrix
```

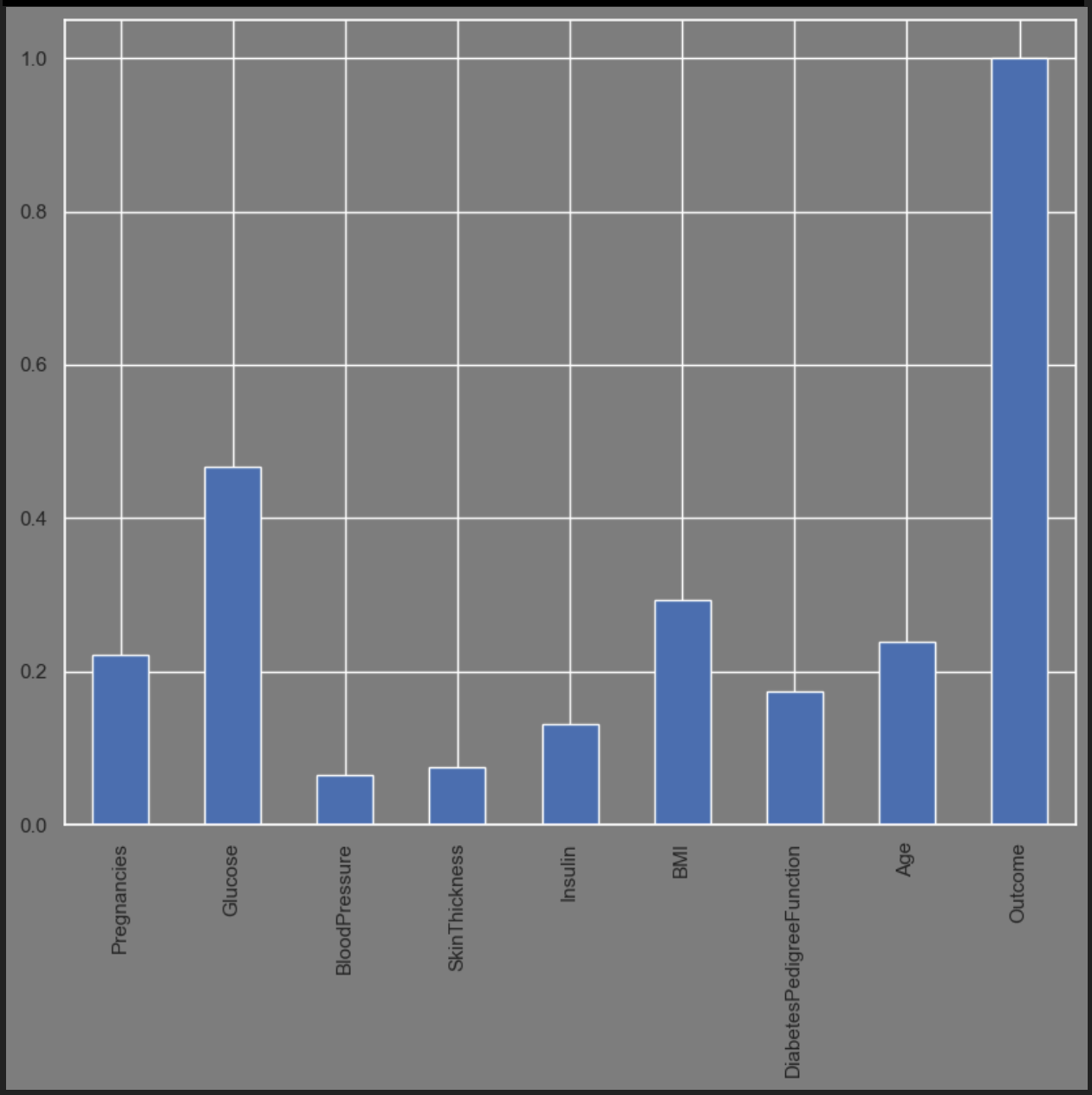
```
corr_matrix = df.corr()

# Set the plot size, color, and font scale
sns.set(font_scale=1)
plt.figure(facecolor='gray', figsize=(10,8))

# Display the matrix
sns.heatmap(corr_matrix, annot=True, fmt='.0', cmap='coolwarm')
plt.show()
```



Creating a correlation matrix of diabetes data with pandas and seaborn.

```
# Set the plot size and color
plt.figure(facecolor='gray', figsize=(10,8))
ax = plt.axes()
```

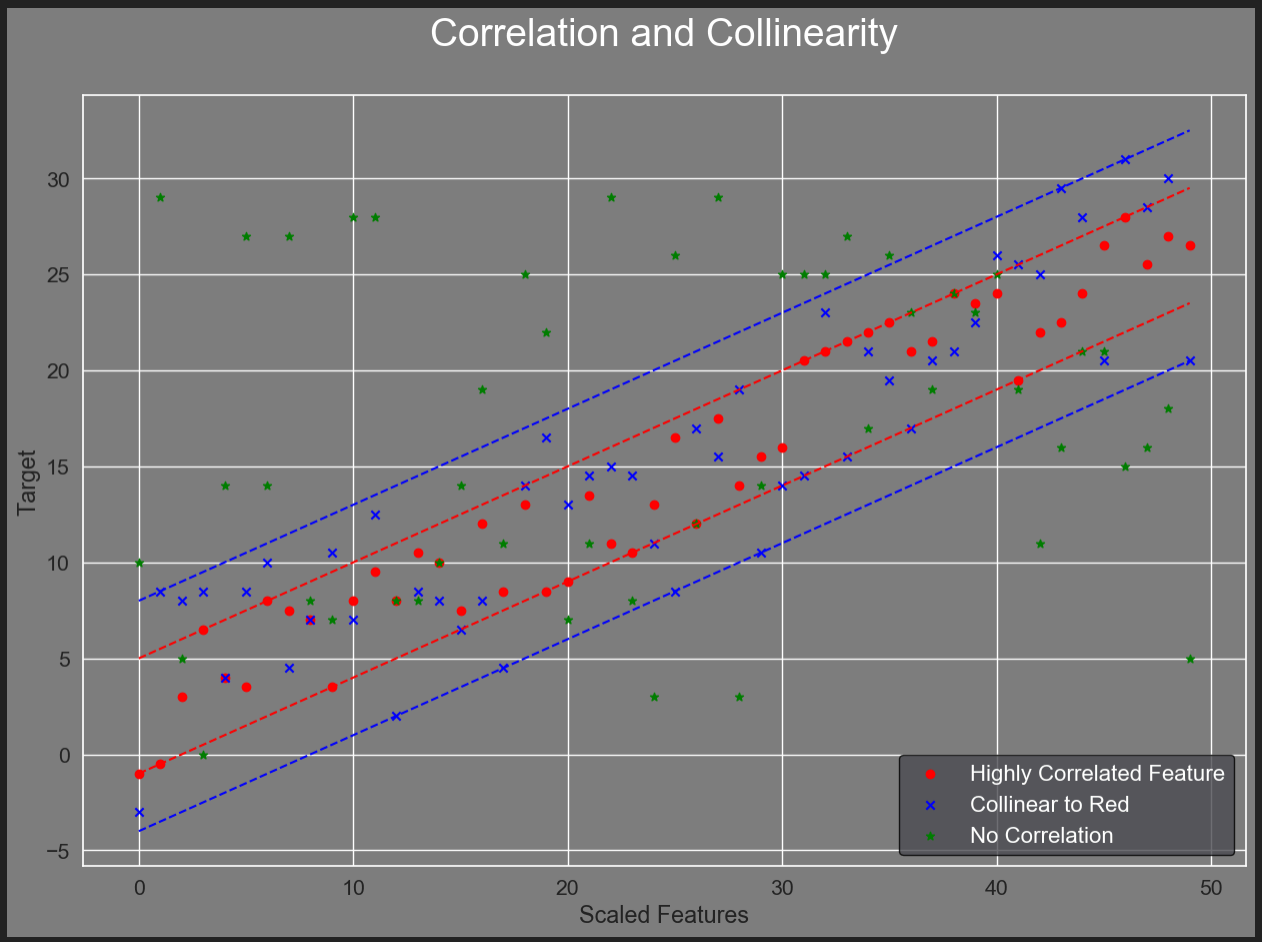
```
ax.set_facecolor("gray")

# Display a plot of Target correlations
target = corr_matrix['Outcome']
target.plot(kind='bar')
plt.show()
```



Generating a bar plot showing each feature's correlation with the target variable.

Feature Interdependence

In machine learning, feature interdependence refers to the degree to which each feature or variable in a dataset is statistically dependent on the other features. When two or more features are highly correlated or dependent on each other, it can indicate multicollinearity. Where correlation refers to the degree of linear association between two variables, collinearity refers to the presence of a high degree of correlation between two or more features in a regression model, and multicollinearity refers to a situation in which three or more features are highly correlated. This can cause problems in machine learning because it violates one of the fundamental assumptions of linear regression, which is that the predictor variables are not correlated with each other. When this assumption is violated, it can be difficult to interpret the coefficients of the model and to make accurate predictions.



In this figure, three features (red, green, blue) are plotted against their accompanying target values. The green feature shows no correlation with the target, and thus is not useful in predicting target values. Blue and red both have some correlation with the target and share a collinear relationship where they would have roughly the same line of best fit. However, the correlation of blue is lower, leading to larger standard errors and a less accurate coefficient.

Multicollinearity can negatively impact many machine learning models, including linear regression, logistic regression, and some forms of decision trees. In linear regression, multicollinearity can cause the coefficients of the model to be unstable and difficult to interpret. It can also result in incorrect estimates of the standard errors of the coefficients, which can affect the statistical significance of the model. In logistic regression, multicollinearity can cause similar problems, such as unstable coefficients and incorrect standard errors.

To address feature interdependence, one approach is to use feature selection techniques to remove redundant or highly correlated features from the dataset. While there are automated tools available to assist in doing this, a simple approach is to again consult the correlation matrix. If two or more features are highly correlated with each other, remove the one that is least correlated with the target. Test the model and repeat the process until the model score either drops or plateaus. By removing these redundant features, the algorithm can focus on the most important features, leading to better performance and more interpretable results. Another approach is to use regularization techniques, such as Lasso or Ridge regression, which can reduce the impact of correlated features on the model coefficients.

Useful links:

- Multicollinearity in Regression

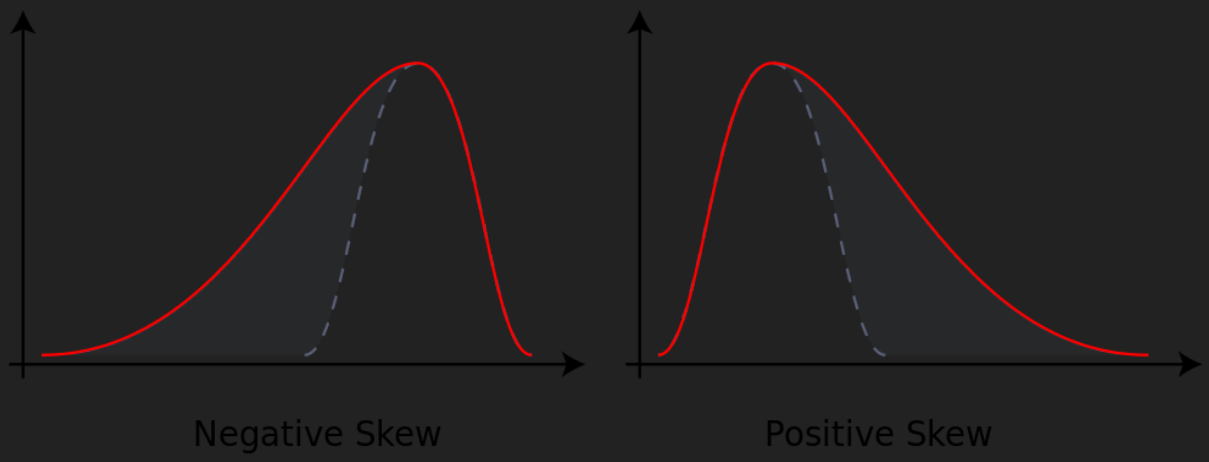
Feature Distributions

There are several basic distributions that can be found in data used for machine learning, and understanding these distributions is important for selecting appropriate models and preprocessing tools. Some common distributions include the normal distribution, the uniform distribution, and the skewed distributions (left and right skewed), and bimodal distributions.

The normal distribution is a bell-shaped distribution that is symmetrical around the mean, with the majority of the data clustered around the mean and fewer data points further away. Many machine learning models assume that the data is normally distributed, and this distribution is important for calculating probabilities and making accurate predictions. Normal distributions will have a mean, median, and mode that are roughly equal to each other.

Skewed distributions can occur when data is not entirely normally distributed, with the direction of the skewness indicating the direction of the longer tail of the distribution. This means that ***right***, or ***positive***, skewed distributions have the bulk of the data to the

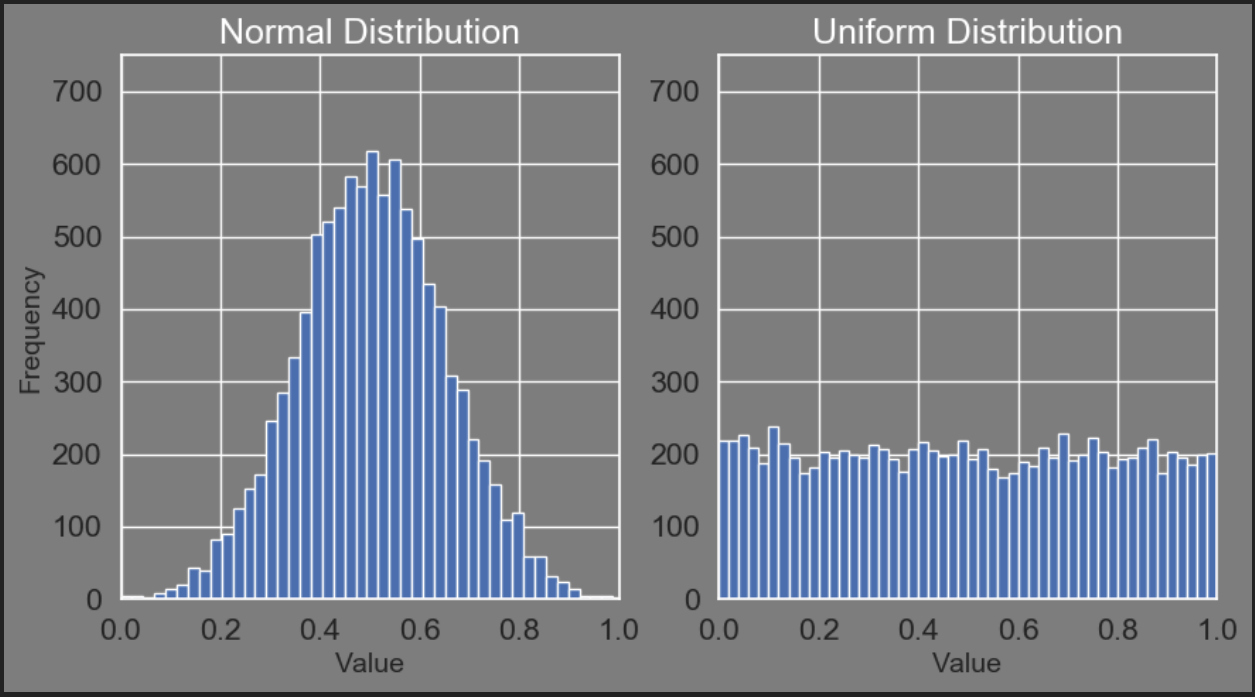
left with a longer tail to the right, or positive direction, while ***left***, or ***negative*** skewed distributions have the bulk of the data to the right with a longer tail to the left. If a distribution appears to be a skewed normal distribution, that is a good indicator there are outliers present in the dataset.



Courtesy of Wikipedia. Used without permission because I am a stud. Note the skew of a distribution is to the direction of the exaggerated tail.

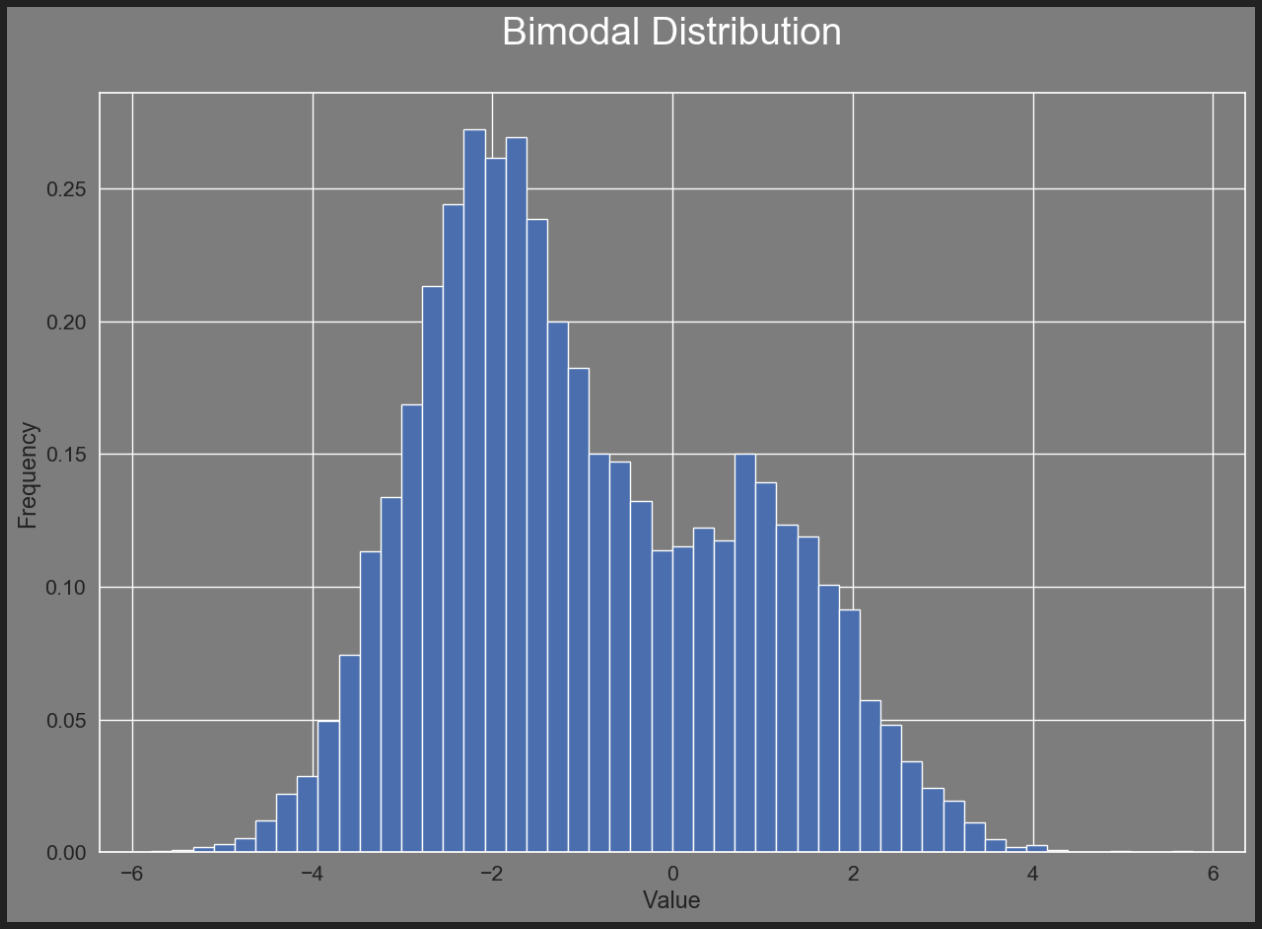
{ PARETO DISTRIBUTION }

The uniform distribution is a flat distribution where all values are equally likely. In a discrete uniform distribution, where the possible values are a finite set of equally likely integers, the probability of finding a data point at any given value is simply $\frac{1}{N}$, where N is the number of possible values.



A comparison of Normal and Uniform Distributions. Uniform Distributions are characteristically "flat" when modeled.

Bimodal distributions are distributions with two distinct peaks, i.e., two modes. The presence of multiple modes in a dataset is a strong indicator that the data contains multiple populations. Alternatively, a bimodal distribution may indicate a measurement or data retrieval error in the dataset.



In supervised models, bimodal distributions can present certain challenges to training and interpreting accurate models. While some models only work under the assumption of unimodal, normally distributed data, others will still require that the relevant feature or features causing the distinct populations be present in the dataset. Identifying the key features that distinguish the two groups and separating them for individual analysis will improve model performance and interpretability.

Bimodal distributions can also cause problems in sampling. During the train/test split, if one of the modes, and therefore populations, dominates the training set, it can result in the models' accuracy being negatively impacted. On the other hand, the model may ignore the smaller population altogether to max out accuracy on the bulk of data. Similarly, if one of the populations dominates the testing set, the model may appear to be more accurate than it really is. This means that any data that is a member of the subpopulation will have consistently incorrect results, even if the model appears to otherwise perform well.

Clustering offers a method of determining what data is in each subpopulation. If the populations can be successfully identified and separated, one approach to handle them is to add a feature distinguishing them in the dataset. Another would be to create separate models for each population.

The simplest way to determine the shape of a data distribution is to start by visualizing it. This can be done in Python using the `matplotlib` library to generate a ***probability density chart*** for each individual feature with ***numeric*** data in the dataset.

```
# Importing our Dependencies
import matplotlib.pyplot as plt

# Generate a Density Plot
plt.hist(df['Feature 1'], density=True)
plt.title("Feature 1 Density Plot")
plt.show()
```

Annotation

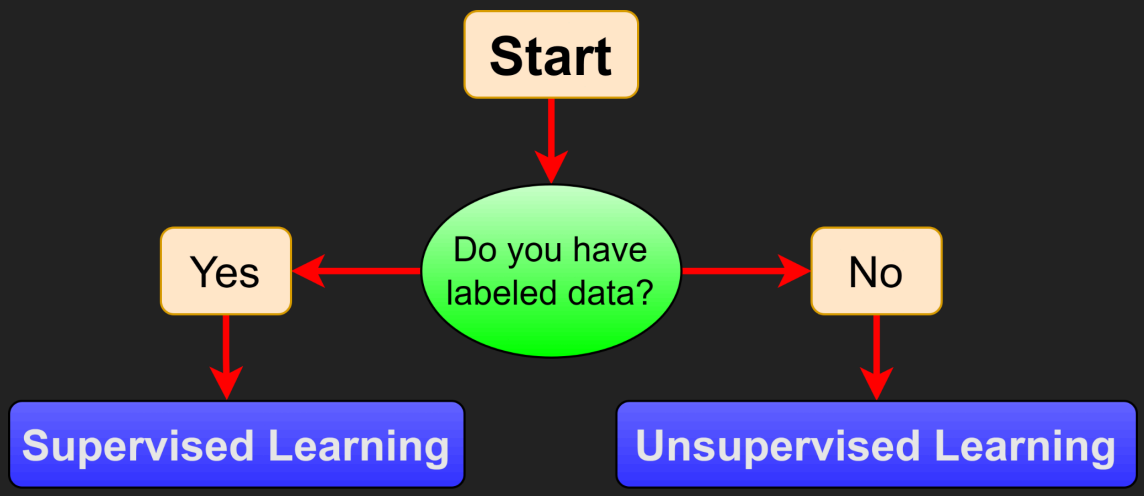
While visualization is an important first step toward understanding the general shape of the data, determining normality usually requires more empirical analysis.
The `scipy.stats.normaltest()` function is a statistical test that determines whether a given sample of data is normally distributed or not. The test works by computing the ***skewness*** (The left vs. right asymmetry) and ***kurtosis*** (The peakedness vs. flatness) of the data and comparing them to what would be expected for a normal distribution.

```
# Importing our Dependencies
from scipy.stats import normaltest

# Performing the normal test
k2, p = normaltest(df['Feature 1'])
alpha = 0.05

# Printing the results
print(f"Feature 1 p-value = {p[0]}")

if p[0] > alpha:
    print("The data is normally distributed.")
else:
    print("The data is not normally distributed.")
```

The `scipy.stats.normaltest()` function computes the skewness and kurtosis of a given sample of data and tests whether the data is normally distributed using a test statistic and p-value. In this example, the p-value is compared against a standard alpha value of 5% (0.05). If the p-value is smaller than the alpha, the null hypothesis of normality is rejected, indicating that the data is not normally distributed. If it exceeds this value, the data is considered normal.

Model Selection

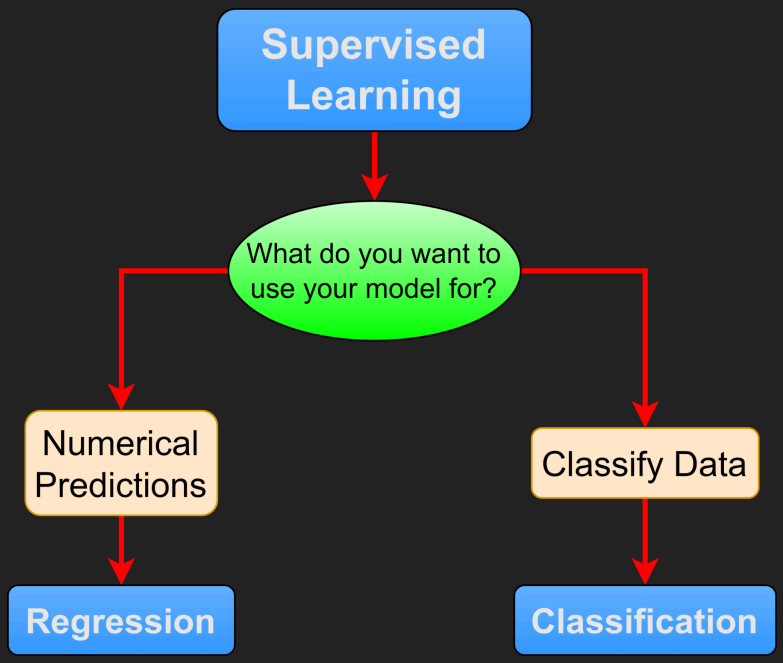
Model Selection is primarily driven by two factors: the kind of data and the kind of question or problem one seeks to use it for.



{ LABELED DATA EXPLANATION }

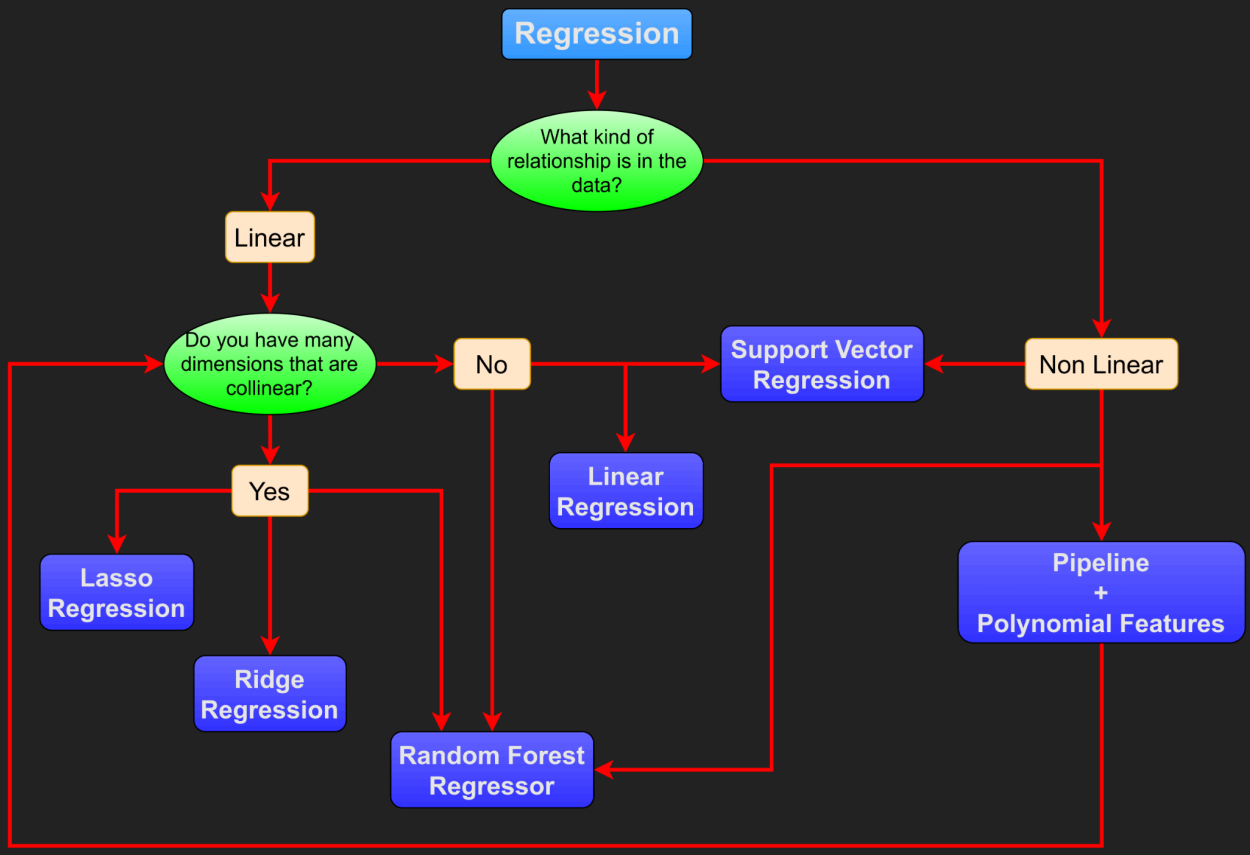
Supervised Machine Learning Models

Is the dataset labeled? Labeled datasets can be used with **Supervised Learning Models** to best answer questions.



Numerical Prediction Models

Is the target both quantitative and continuous? I.E., is the prediction intended to be any real number? Models to consider are:



- **Linear Regression**

  Linear regression is a popular machine learning algorithm used for predicting continuous numerical values based on a set of input features. The basic idea behind linear regression is to fit a straight line to the data that minimizes the difference between the predicted values and the actual values.
  Linear regression can be most easily understood by the following equation:

$$y = mx + b + e$$

  Where:

  - y is the target variable
  - x is the input feature
  - m is the slope coefficient of the line
  - b is the y-intercept
  - e is the error term.

The goal of linear regression is to find the values of m and b that minimize the sum of the squared errors between the predicted values and the actual values. There are two main types of linear regression: simple linear regression and multiple linear regression. Simple linear regression involves only one input feature, while multiple linear regression involves multiple input features.

Here are some factors to consider when deciding whether to use linear regression over other regression types:

- **Linearity:** Linear regression assumes a linear relationship between the input features and the target variable. If the relationship is not linear, linear regression on its own may not be the best choice.
- **Number of input features:** Linear regression works well when there are a small number of input features. As the number of input features increases, linear regression may become less effective. In such cases, feature selection techniques may be required, or other regression types such as Lasso or Ridge regression, which can handle high-dimensional data, may be more suitable.
- **Outliers:** Linear regression is sensitive to outliers, which can greatly affect the model's performance.
- **Non-normality:** Linear regression assumes that the data is normally distributed.
- **Interpretability:** Linear regression is a simple and transparent model, which makes it easy to interpret and understand the relationships between the input features and the target variable.

Useful links:

- sklearn.linear_model.LinearRegression — scikit-learn 1.2.1 documentation

- **Lasso Regression**
  {{ TEXT }}

  Useful links:
  - _

- **Ridge Regression**
  {{ TEXT }}

  Useful links:

- _

- **Random Forest Regressor**
  The random forest regressor works essentially the same as the random forest classifier under the hood, but is used for regression tasks, where the goal is to predict a continuous numeric output based on any multitude of types of input features. Random forest regression works by building an ensemble of decision trees, where each tree is built using a random subset of the features and an accompanying random subset of the training data. The final prediction is then made by averaging the predictions of all the individual trees in the forest.

  Here are some indicators for why random forest regression might be a good choice over other regression types:

  - **Non-Linearity:** When the relationships between the input and output variables are non-linear or the data contains complex interactions between the input variables, which can be better captured by decision trees.
  - **Number of input features:** If the data contains a large number of input variables, and it is difficult to identify which ones are the most important for making accurate predictions. Random forest regression can automatically select the most important variables.
  - **Robustness:** If the data contains missing values, unusual distributions, or outliers, these can be handled by random forest regression while requiring minimal data preprocessing. Random forest regression is also generally robust to a lack of scaling.
  - **Mixed data types:** Random forest regression is an ideal option when the data contains a mixture of categorical and continuous input variables, which can be handled by random forest regression without requiring one-hot encoding of the categorical variables.

  Useful links:
  - sklearn.ensemble.RandomForestRegressor — scikit-learn 1.2.1 documentation

- **Support Vector Regression**
  {{ TEXT }}

  Useful links:
  - _

Classification Models

Is the target a discrete value over a definite range or a qualitative data point? If so, the target is considered to be **classed**, and classifier models can be used to predict these classes.

- ### Random Forest Classifier

  As stated above, random forest classification and regression work in essentially the same way under the hood. Overall, random forest classification is an extremely powerful machine learning model that tends to provide good performance even with minimal hyperparameter tuning, making it a convenient starting point. In addition to the previous features and factors, consider the following when choosing whether or not to use random forest classification over other models.

  - **Handling imbalanced datasets:** Random forest can handle imbalanced datasets by adjusting class weights or using techniques like bootstrapping and balancing classes during training.
  - **Interpreting results:** Random forest provides insights into feature importance, allowing interpretation of the model and understanding the factors driving the classification decisions.
  - **Resistance to overfitting:** Random forest combines multiple decision trees, reducing overfitting and improving generalization performance compared to individual decision trees.
  - **Scalability:** Random forest can handle large datasets efficiently, thanks to its parallelization capabilities and the ability to train on subsets of data.

  Useful links:

  - _

- ### Logistic Regression

  {{ TEXT }}

  Useful links:

  - _

- ### K Nearest Neighbors

  {{ TEXT }}

  Useful links:

  - _

- **Support Vector Machine**
  {{ TEXT }}

  Useful links:
  - _

- **Naive Bayes**
  {{ TEXT }}

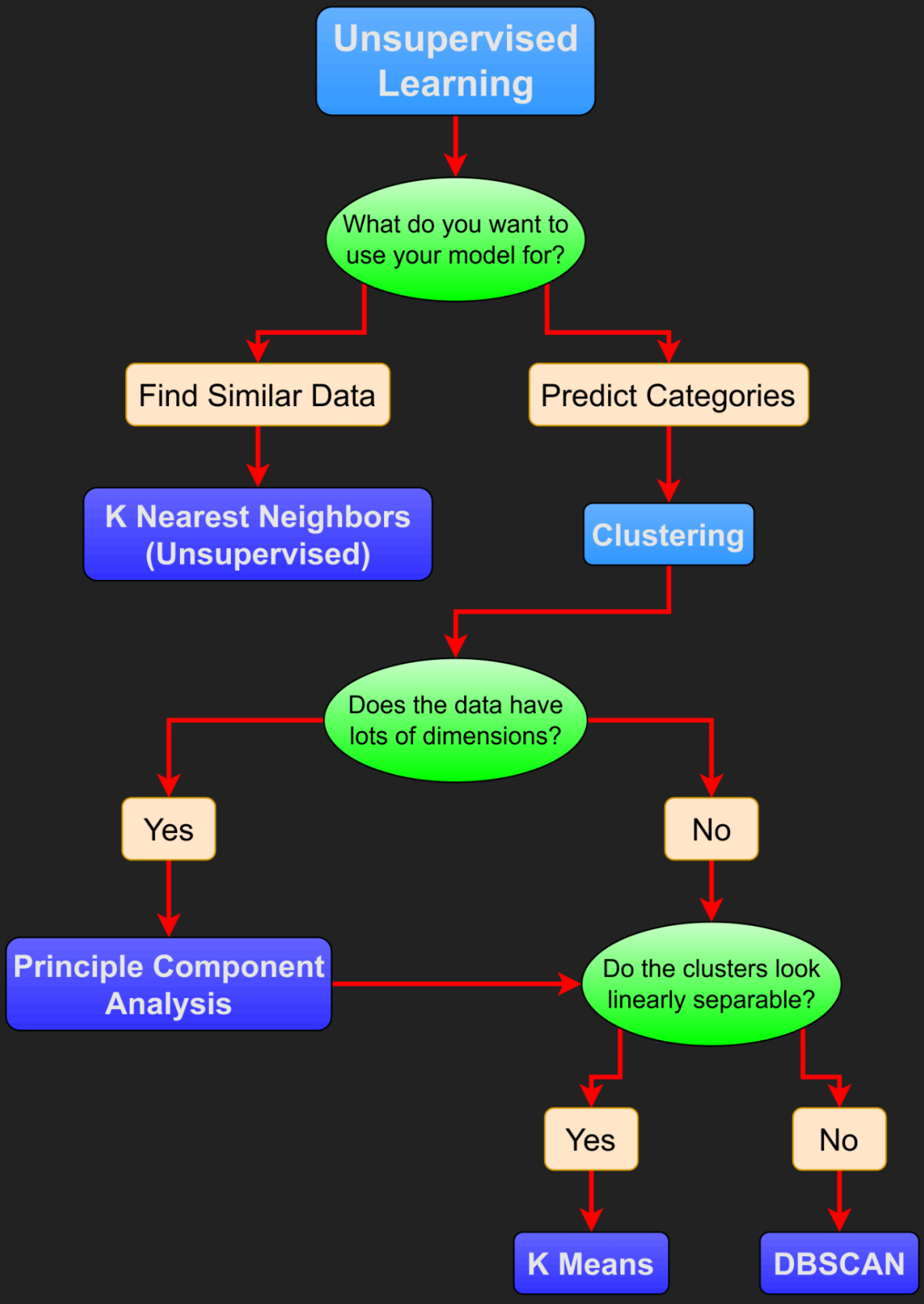
  Useful links:
  - _

- **Neural Networks**
  {{ TEXT }}

  Useful links:
  -

Unsupervised Machine Learning Models

Is the dataset unlabeled? Use **Unsupervised Learning Models** to explore the data.


Unsupervised Learning
What do you want to use your model for?
Find Similar Data
Predict Categories
K Nearest Neighbors (Unsupervised)
Clustering
Does the data have lots of dimensions?
Yes
No
Principle Component Analysis
Do the clusters look linearly separable?
Yes
No
K Means
DBSCAN

Clustering Models

{{ TEXT }}

- **K Means**
  {{ TEXT }}

  Useful links:
  - sklearn.cluster.KMeans — scikit-learn 1.1.2 documentation
  - 2.3. Clustering — scikit-learn 1.1.2 documentation

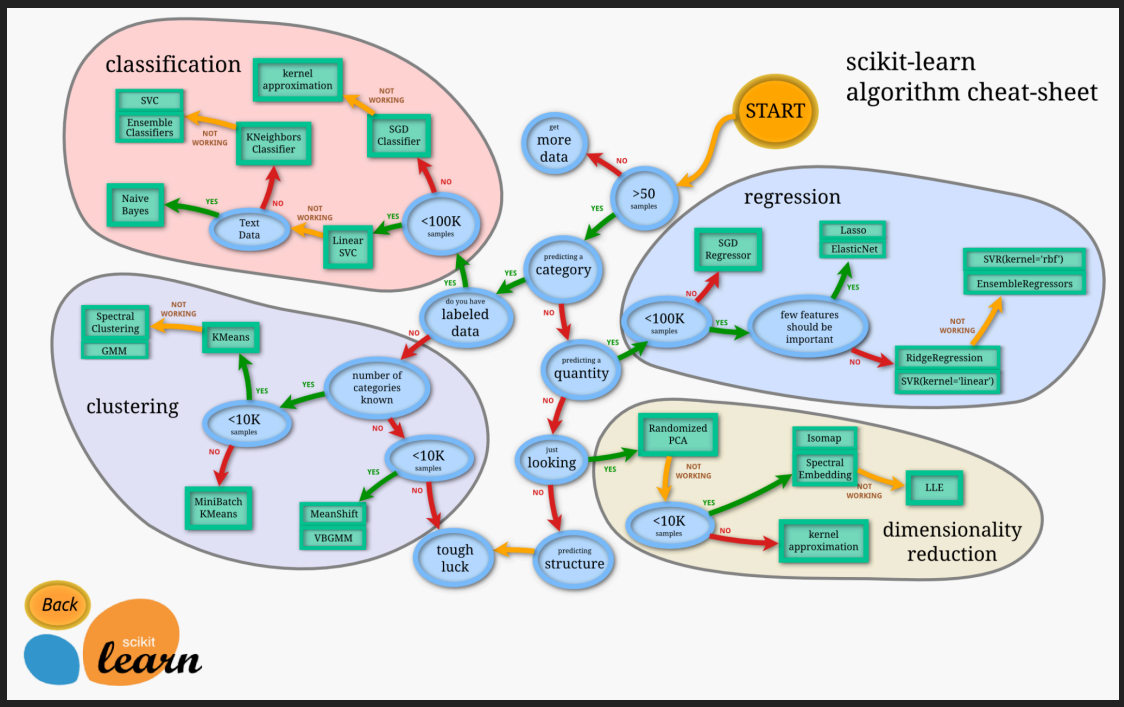
- **DBSCAN**
  {{ TEXT }}

  Useful links:
  -

Find Similar Data Points

- **K Nearest Neighbors**
  The unsupervised implementation of the Nearest Neighbors algorithm. Allows the user to find similar data points to an input data point.

  Useful links:
  - sklearn.neighbors.NearestNeighbors — scikit-learn 1.1.2 documentation
  - 1.6. Nearest Neighbors — scikit-learn 1.1.2 documentation



Preprocessing Data For The Model

After selecting models to try, it is necessary to prepare the data for use with that particular model.

Scaling

Depending on which model you choose to use and what your data looks like, you may need to scale your data to get better results. There are several different scalers on scikit learn to choose from, and some may produce better results with the dataset and selected machine learning model than others. Consider investigating how the following might affect the data:

- **StandardScaler**
  The StandardScaler is used to scale and center a dataset. The standard scaler is a commonly used scaling technique in machine learning, and is especially useful when the features of the dataset have different scales, as it can help to improve the performance of some machine learning algorithms.

  The StandardScaler works by subtracting the mean value of each feature from the data point and dividing it by the standard deviation of the feature, essentially converting each data point within a feature to its z-score.

$$z = \frac{x - \bar{x}}{\sigma}$$

Where:

- z is the scaled data point
- x is the unscaled data point
- $\bar{x}$ is the mean value of the feature
- σ is the standard deviation of the feature

This process produces a feature with a mean of zero and a standard deviation of one, thus standardizing the range of each feature. This makes it easier for the algorithm to learn from the data, as it can be sensitive to the scale of the input features.

Useful links:

- sklearn.preprocessing.StandardScaler — scikit-learn 1.2.1 documentation

- **MinMaxScaler**
  {{ TEXT }}

  Useful links:
  -

- **MaxAbsScaler**
  {{ TEXT }}

  Useful links:
  -

- **RobustScaler**
  {{ TEXT }}

  Useful links:
  -

Feature Engineering

Many models will benefit from some degree of feature engineering. This is the process of developing new features from the original data. Creating this new data typically requires some level of domain knowledge.

Useful links:

- What is Feature Engineering — Importance, Tools and Techniques for Machine Learning | by Harshil Patel | Towards Data Science

Feature Selection

Feature selection really happens throughout the whole process (from data cleaning to model tuning). This is simply deciding what features are most likely to be useful to include in training the model and dropping those that are not. It is useful to reference back to the EDA step here and consider data relationships like feature interdependence and feature relevance.

Data Leakage

Data leakage in machine learning is a rather embarrassing condition where data from outside the training set accidentally seeps into the ~~pants~~ model, contaminating the results and rendering them useless. Avoiding data leakage is a critical step to prevent the model from feeling foolish, a bit warm, or like it needs a diaper, but with just a bit of preparation and careful attention, it will be easy to avoid any such unwanted accidents.

Data leakage mostly happens in feature engineering, categorical encoding, scaling, reshaping, and data transformation. A good rule of thumb is: if it uses `.fit()`, remember to `.split()`. In other words, remember to do a train/test split of the data beforehand and use `.fit()` ONLY on the training data, and then `.transform()` the training data and test data separately. It is a best practice to also export these objects for re-use during your deployment.

Useful links:

- Data Leakage in Machine Learning

Categorical Encoding

Once your data is almost ready to be past to a model, you need to encode your categorical variables. First, look at the unique values of each categorical column. If a column has 200 unique values, this will turn into 200 new columns just for this feature. Do you really want to keep this feature? (hint: feature selection, again)

- **OneHotEncoder**

- **LabelEncoder**

Reshaping Data

{ TEXT }

Dimensionality Reduction

If you decided to keep that feature, maybe it would make sense to look at some dimensionality reduction techniques such as PCA. Remember that the downside of this is the lack of interpretability

How To Use Each Model

Reference your activities you lazy student, we are not going to write that out for you! Also consider checking out the official docs for your model at the scikit-learn user guide.

Selecting Model Evaluation Metrics

I know we always used the '.score' to evaluate our models in class. For most classifiers, the '.score' refers to the accuracy. For regressors, the '.score' refers to the R^2. Here is the issue: R^2 gets bigger (closer to 1) the more predictors you have in your model, that means that your model may be very bad, but the R^2 would make it look like a good model if you have enough predictors. Something similar but different happens with accuracy, if your dataset is highly imbalanced, your accuracy will tend to be the ratio of the imbalance. In other words: if you have 990 zeros and 10 ones, your accuracy will be 99% because your model will learn that by predicting zeros 100% of the time, the accuracy will be extremely good. To avoid this issues, we need to look deeper into what our goal with the prediction is. For regressions, RMSE is a popular metric, but again, it may not fit your needs. For classifications, you may want to try adjusted R^2, which takes into consideration the number of predictors you are using, or ROC score, or brier score, or F1 score, or precision/recall score, or… you get the idea. I just would personally never recommend using R^2 or accuracy unless you know all the downfalls and know your data very well

Regression Metrics

{ TEXT }

- **R^2**
  { TEXT }

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - f(x_i))^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

- **Adjusted R^2**
  { TEXT }
- **MSE**
  { TEXT }
- **RMSE**
  { TEXT }

Classification Metrics

{ TEXT }

- **The Confusion Matrix**
  { TEXT }
- **Accuracy**
  { TEXT }

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**
  { TEXT }

$$Precision = \frac{TP}{TP + FP}$$

- **Recall**
  { TEXT }

$$Recall = \frac{TP}{TP + FN}$$

- **F1**
  { TEXT }
- **Balanced Accuracy**
  { TEXT }
- **ROC**
  { TEXT }

Model Tuning

You can't tune your model if you don't have the right metric, so go back to section 7. Once you know what your metric is, or metrics, then we can start thinking about model tuning. Model tuning is performed using search spaces methods, the most common are GridSearchCV and RandomizedSearchCV. These are very simple to use, just know what parameters you may want to tweak to improve your metric scores.

GridSearchCV

{ TEXT }

RandomizedSearchCV

{ TEXT }

Keras-Tuner

{ TEXT }

Retrain Best Model(s) With All The Data

Once you have selected the best models for your project, retrain them using all your data (this means that you will 're-fit' all your encoders, transformers, scalers, etc as well). This is because the more data, the better, and you no longer need to evaluate your model. Make sure you use the same parameters that gave you the best results during your model tuning.

Model Deployment

{ TEXT }

Pipelines

{ TEXT }

Saving the Model

{ TEXT }

Loading the Model

{ TEXT }

Modularization

{ TEXT }

Additional Resources

The following is an unstructured list of useful links, courtesy of Thomas Raczkowski:

Machine Learning/Optimization

- 3Blue1Brown “What is a neural network” - https://www.youtube.com/watch?v=aircAruvnKk&t=3s - I know that I recommended this guy already, but if you haven’t watched this yet definitely take the time to check out this series on NNs. His stuff is top-tier for building yoru calculus and linear algebra understanding). 3Blue1Brown is a hero and has helped so many in the linear algebra/advanced math world. Amazing visuals that really help you get the point across.
- “Machine Learning From Scratch: Part 1" - https://towardsdatascience.com/machine-learning-from-scratch-part-1-76603dececa6 - This post on “Towards Data Science” is a 4-parter on all of the major pieces of machine learning. It talks you up from the fundamentals (things we discussed early on in the class, lists, collections, etc.) through classifiers, regression, etc. I would say this is a good read if you feel like you want to bring everything we’ve learned throughout class together. NOTE: Parts 2-4 are

immediately available at the top of part 1. While there is a lot to read here it is a good one!

- Machine Learning is Fun - https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471 - This is a great (if slightly old) writeup on general machine learning and provides a lot of good, general insight on it.
- 51 Essential Machine Learning Interview Questions - Springboard - https://www.springboard.com/blog/machine-learning-interview-questions/ - This list of questions is a pretty good place to hang around for a bit so that you can get a feel for what concepts are important in the ML space. Sure there are plenty of models and techniques to learn from, but these questions give you a feel for the very high-level concepts to look out for.
- Bias-Variance Tradeoff - https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html - So I explained ML as being this constant battle between over-fitting and under-fitting. Here is the concept with a great deal of depth. Feel free to either check out the lecture or just the notes, but this can help you get an understanding for what I meant when I was talking about over and under-fitting. The Bias-Variance tradeoff is an incredibly important concept in ML.
- What is the Bias-Variance Tradeoff? - https://elitedatascience.com/bias-variance-tradeoff - This is a much more approachable explanation of the bias-variance tradeoff than the previous one. The infographic really helps to show the concepts in a way that may be more clear than the equations in the previous link. Definitely check this one out if you get a chance, like I said, the bias-variance tradeoff is one of the more important concepts in ML and it is sure to follow you around.
- Curse of Dimensionality - https://builtin.com/data-science/curse-dimensionality - We have talked about fitting models to both narrow and wide datasets, here are some issues that you may run into with data of very high dimension, as you read through this think about just how much our n-dimensional space grows each time we add a new feature/column into the mix. The curse of dimensionality is similar to the bias-variance tradeoff in that it is a very important ML concept that you should always keep in the back of your mind.
- An In-Depth Guide to How Recommender Systems Work - https://builtin.com/data-science/recommender-systems - Check out this article for quite a bit of information on recommender systems. There are some visuals that help along the way but this is quite a long article that you can get a lot out of. It walks through specific types of recommenders used by top companies (Amazon, Facebook, Youtube, Netflix) in different situations.

- An Introduction to Statistical Learning - https://www.statlearning.com/ - I believe that the ISLR is freely available online at this point. The Introduction to Statistical Learning is a fantastic textbook that covers many of the models that we've discussed in class at a mid-late college textbook level. I leaned on this book quite a bit in the bootcamp I was in as a student and have also been recommended this book by peers in the working world. It's a great one and best of all, it's free! You can't go wrong with this if you want to really sit down and peel apart what we've discussed in class. You can spend months on this book if you want. Hugely useful! Lots of good visuals too spread throughout these pages. Also keep an eye out for the ESLR (Elements of Statistical Learning) which is sort of the older sibling of this book. I believe that is also available online if you look around Google a bit.
- Seasonal Arima Model - https://otexts.com/fpp2/seasonal-arima.html - This will give you some quick insight into how Arima models work. The code here is R, but it would be quite similar to employ Arima in Python. This should give you a solid primer on some of the methods and pit-falls you'll run into with Arima. If you poke around the rest of this page as well, you'll find that there is a depth on forecasting methods of all kinds, and talks through important concepts such as…
- Time Series Decomposition - https://otexts.com/fpp2/decomposition.html - A very important piece about time-series models is how they peel the various components of a trend like the layers of an onion. Think about Trying to make a phone-call on 12:01 AM on Wednesday January 1st of some year, the situation is unique. Wednesdays on their own don't necessarily result in an outsized amount of calls, 12:01 is not necessarily the busiest time of day for phonecalls, but because you're putting that together with a longer trend (New Years Eve which happens every ~365 days), you end up with the possibility of many calls getting dropped. The idea here is that you can learn how to break down your analysis of time into these components.
- How to Create an ARIMA Model for Time Series Forecasting in Python - https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/ - This is some more ML specific code on time-series forecasting with ARIMA.
- Facebook Prophet documentation - https://facebook.github.io/prophet/docs/quick_start.html - This is another method of handling time-series data and may be favorable compared to ARIMA if trained properly, I'm not familiar with it in my day-to-day but I've only ever heard good things about this library.
- Edureka - Decision Trees - https://www.edureka.co/blog/decision-trees/ - This blogpost is full of great visuals about decision trees, and I would say that that's the biggest strength here, there are some good broad strokes and important words to cover. Keep an eye out for "entropy" and "information gain", these are

topics that I briefly talked about during lecture and if you want to learn about the intricacies of decision you should search around some more on those topics. Tree-based models are incredibly interesting and I think they're a lot of fun.

- An Introduction to Boosted Trees - https://xgboost.readthedocs.io/en/latest/tutorials/model.html - This is a link to the XGBoost documentation. This is a technique that is also tree-based, but the way the trees are built up is slightly different than random forests which we talked about in class. XGBoost was the technique I ended up using in my final project and it used to be the algorithm to beat for a little while in the ML competition space. Check it out if you have the bandwidth! The page I link to talks about the process of additive training and building the trees up in series as well as in parallel. There is some really hairy math on this page, but there are also some great visuals and a really solid description as well, try to gloss over the math unless you feel really confident in your abilities, but try to get an understanding why we would build trees in this way. This was the technique I mentioned that builds trees in series rather than in parallel (like Random Forest), so you use the result of old trees and then iterate upon them.
- Types of Clustering Methods: Overview and Quick Start R Code - https://www.datanovia.com/en/blog/types-of-clustering-methods-overview-and-quick-start-r-code/ - Many of the more interesting and conceptual articles on clustering and ML in general are written with R as the language they code in, but don't let that scare you off. Try to glean what the clustering concepts are here and there will likely be ways to implement them in Python as well. I wouldn't worry much if the concepts in this article are a little bit more foreign, but you can understand a little bit more about Kmeans clustering as a result.
- A Step-By-Step Explanation of Principal Component Analysis - https://builtin.com/data-science/step-step-explanation-principal-component-analysis - PCA is a technique for dimensionality reduction, it is some linear algebra that sort of understands the shape of our data and almost narrows space around it. I think that PCA is quite confusing and tough to conceptualize, but there is a visualization about halfway through this article that shows how it works quite nicely (look out for the spinning red lines and the blue dots). If you've ever taken a linear algebra class, some of that will likely come rushing back.
- Selecting the Number of Clusters with Silhouette Analysis on Kmeans Clustering - https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html - Silhouette analysis is the process of creating several of these "sail" graphs to get a better understanding of your clusters and it will help you optimize the number of clusters you choose.

- K-means Cluster Analysis - https://uc-r.github.io/kmeans_clustering - This is another example of a really well written article in R about k-means clustering. It really allows you to get a great idea of how these clusters work, how to best choose your optimal number of clusters (using the elbow method), and goes into the silhouette method a little bit as well. Though the code here is in R, I would recommend that you find some of the Python code that has been built up for Kmeans (Sklearn's documentation will help you out a lot) and try to see if you can build it while you read through this article.
- K-Nearest Neighbors - https://pythonbasics.org/k-nearest-neighbors/ - This is just a brief writeup on KNN and how to implement it in Python. KNN is one of the simpler algorithms we've worked with so I just wanted to include something quick on it as a bit of a refresher.
- Classifying Data using Support Vector Machines in Python - https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-python/ - This is a slightly longer read about SVMs in Python. As I mentioned in class, SVM almost "teases" space into our data so that it can classify using the optimal hyperplane. Thinking about SVM in many dimensions is pretty mind-boggling, this is another favorite of mine conceptually alongside tree-based models.
- Comparing Machine Learning Methods - https://pythondata.com/comparing-machine-learning-methods/ - This article talks a little bit about Kfolds cross-validation (K-Folds is similar to a train-test split but it essentially iterates through several train-test splits. K-Folds is a really useful technique for comparing models as it allows us to compare models in a way that is a bit more advanced than the standard train-test split that we've talked about in class. You'll find that you can choose between a regular train-test split and K-Folds for model comparison in many circumstances.
- How do Convolutional Layers work in Deep Learning Neural Networks? - https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/ - Convolutional layers and Convolutional Neural Networks are very often used in image classification. This is pretty sparse on visualization, but if you read through it it should give you a good bit of detail on why you'd want to use these layers.
- How to Classify Photos of Dogs and Cats (with 97% accuracy) - https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/ - This is an insanely in-depth read on the techniques used to build a really solid image classifier for dogs and cats. From what I'm seeing every single line of code is available and you can really get a good idea for what the original creator wanted to do at each step. They talk about image pre-processing in batches (something that people may forget is a part of

the process) and just about anything else you'd need to build a "basic" image classifier.

- A comparison of grid Search and Randomized Search using Scikit learn - https://blog.usejournal.com/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85 - This comparison between grid and randomized search will give you a better understanding of what exactly grid search is doing as well as some issues that may come with that territory. As I've discussed, grid search is a time-consuming (and processor-intensive) process that allows you to train up a ton of models with different hyperparameters. Randomized Search is a useful alternative and may help you save on time in some cases, and I just want to point out here that grid search is typically going to be one of your last steps in tweaking a model. You typically won't go from a model that classifiers correctly 25% of the time to one that works 75% of the time just by searching the hyper-parameter space (unless you started off with some purposefully bad initial hyperparameters), you should rely on grid search mainly to eke out the last bits of performance from your model, and if your data is not conducive to an ML model, don't expect any magic at that stage.
- Grid Searching for Good Wine - https://towardsdatascience.com/grid-searching-for-good-wine-43cdfeadee6c - A quick example of grid search being used to look at data on some various wines.
- Metalearner Algorithm - H2O AutoML- http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/metalearner_algorithm.html - This talks a little bit about the H2O AutoML metalearner algorithm that is available. This is sort of like a mega grid search that H2O uses for their AutoML library. I would also point out that this is a great site to poke around and I would recommend that you set up a virtual environment for automl, get it, and throw it at some data just to get a feel for how it works. They have really done all they can to make ML training trivial, tools like this one are why it has become more important that you can interpret an ML model if you are a data scientist rather than just build one from scratch. This is a fascinating library.
- A video for H2O AutoML - https://www.youtube.com/watch?v=42Oo8TOl85I - This is a mind-blowing video about what AutoML can do! I've watched this one many times over the last few years, it is just fascinating to me how easy it is to train up hundreds of models with little to no thought process.
- An Overview of Gradient Descent Optimization Algorithms - https://ruder.io/optimizing-gradient-descent/ - I'm not expecting you to read every last line of this one, but it can give you a good idea on different gradient descent algorithms (or how they work in general if you're not too comfortable with that just yet). It talks about so many different pieces of gradient descent that are important. I would definitely gloss over the really math-heavy parts of this article

(unless your background lends itself to you really digging in) for now and mainly use it to get an understanding for what gradient descent is doing. There is a little bit in here about Adam as well from the neural networks class.

- Introduction to the Deep Learning Library Tensorflow - https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/#:~:text=TensorFlow%20is%20a%20Python%20library,built%20on%20top%20of%20TensorFlow. - This is just a high level overview of what Tensorflow is, where you can get it, and it links out to some interesting tutorials. Keep in mind that we used Tensorflow behind the scenes of Keras in class.
- One-Hot encoding vs. Label Encoding using Sci-Kit Learn - https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/#:~:text=One%2DHot%20Encoding%20is%20another,process%20of%20creating%20dummy%20variables. - I've recommended some stuff from analyticsvidhya in the past, here they just go into a bit of detail about the key differences between one-hot encoding and label encoding. This should hopefully be a refresher for most of you.
- Loss and Loss Functions for Training Deep Learning Neural Networks - https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/ - This is a fairly solid overview on some different types of loss functions and which loss functions some of the more common ML methods use. There is a very important subheading here: "Neural Network Learning as Optimization" which gets the point across about how a lot of things we've talked about in the ML space are basically just applied optimization.
- A Gentle Introduction to Cross-Entropy for Machine Learning - https://machinelearningmastery.com/cross-entropy-for-machine-learning/ - I briefly talked about cross-entropy and this article/code samples should help you get a bit of a better understanding about that particular type of loss function.
- Understanding Categorical Cross-Entropy Loss, […] Focal Loss, and All Those Confusing Names - https://gombru.github.io/2018/05/23/cross_entropy_loss/ - As stated in the title, this article will help you make heads and tails of several different loss functions. I briefly covered categorical cross-entropy and Softmax in class. You can spend a little bit of time in here reading and you'll end up with way more context than what I described in class.
- The Vanishing Gradient Problem - https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484 - The Vanishing gradient is an issue that has plagued the Neural Network space for years, but recently mathematicians have come up with ways to work around that issue. This is way beyond the scope of class but I figure some of you may find it interesting. I've heard the "solution" of the vanishing gradient descent

problem credited as the reason why we can build such robust neural networks with relative ease.

- What is the Softmax Function? - https://deepai.org/machine-learning-glossary-and-terms/softmax-layer - There were some moments during the Neural Networks lecture that I didn't get into much detail on such as the softmax function. If you recall, we used it to build the output layer of our neural nets, here is some more context on that.
- Yannic Kilcher Youtube - https://www.youtube.com/c/YannicKilcher/videos - This is a guy I only stumbled upon recently, but he is an AI/ML researcher with some really insane videos. He reviews papers that are coming out currently so he's always talking about the bleeding-edge of neural network tech. If you pick something random of his and watch through a bit you may end up gaining from it. His stuff is definitely some of the most challenging I've seen, another incredible youtuber teaching some crazy stuff.
- Matplotlib Contour Demo - https://matplotlib.org/stable/gallery/images_contours_and_fields/contour_demo.html - This is the technique that was used in class to build the plots for the SVM section. Contour plots are a really useful technique for low-dimensional data, or you can use it on subsets of features to see how they compare.
-
- Check out "Cross-Validated", "Analytics Vidhya", "Statistics How To", and "Stack Exchange" as a few general sites with a ton of info on stats/ML. Just add one of those site names to an ML-related search-term and you'll almost definitely find something.

Unannotated ML/AI links

(I apologize that I can't give a primer/overview for each article but still wanted to share…) - Thomas

- Accuracy, Precision, Recall, or F1? - https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9
- Difference between PCA and feature selection in ML? - https://www.quora.com/What-is-the-difference-between-principal-component-analysis-PCA-and-feature-selection-in-machine-learning-Is-PCA-a-means-of-feature-selection
- Support Vector Machine - Introduction to Machine Learning Algorithms - https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
- Wikipedia - Ordinary Least Squares Method - https://en.wikipedia.org/wiki/Ordinary_least_squares

- Wikipedia - Mean Squared Error - https://en.wikipedia.org/wiki/Mean_squared_error
- Lostic Regression Read The Docs/Cheatsheet - https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
- Dimensionality Reduction and visualization using PCA (Credit Corey) - https://medium.com/@ashwin8april/dimensionality-reduction-and-visualization-using-pca-principal-component-analysis-8489b46c2ae0
- Ridge and Lasso Regression: L1 and L2 Regularization - https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b
- Plotting Decision Surface For Classification ML Algorithms - https://www.analyticsvidhya.com/blog/2020/08/plotting-decision-surface-for-classification-machine-learning-algorithms/
- A Complete Tutorial on Ridge and Lasso Regression in Python - https://www.analyticsvidhya.com/blog/2016/01/ridge-lasso-regression-python-complete-tutorial/
- How to fix vanishing gradients problem using ReLU - https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/
- Keras Dense Layer Explained for Beginners - https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/
- Activation Functions in Neural Networks - https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
- Introduction to SoftMax for Neural Networks - https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/
- Introduction to Cross-Entropy for Machine Learning - https://machinelearningmastery.com/cross-entropy-for-machine-learning/
- What is the role of bias in Neural Networks? - https://stackoverflow.com/questions/2480650/what-is-the-role-of-the-bias-in-neural-networks
- Elbow Method in Supervised Machine Learning (Optimal K Value) - I got lots of questions on this concept - https://stackoverflow.com/questions/2480650/what-is-the-role-of-the-bias-in-neural-networks
- Sklearn - RandomForestClassifier Documentation - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

- “The Kernel Trick” - SVM - https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html
- Wikipedia - Hyperparemeter (Machine Learning) - https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)
- Feature Importance - What’s in a name? - https://medium.com/bigdatarepublic/feature-importance-whats-in-a-name-79532e59eea3
- Probability concepts explained: Maximum Likelihood Estimator - https://towardsdatascience.com/probability-concepts-explained-maximum-likelihood-estimation-c7b4342fdbb1
- 4 Types of Distance Metrics in Machine Learning - https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/
- Eigenfaces: Recovering Faces from Ghosts - https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184
- What is the relationship between Loss and Accuracy in Deep Learning? - https://datascience.stackexchange.com/questions/42599/what-is-the-relationship-between-the-accuracy-and-the-loss-in-deep-learning
- One-Hot vs Label Encoding - https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/
- Researchgate - Visualization of different models (This is a nice example to try to visualize several models) - https://www.researchgate.net/figure/sualization-of-different-models-From-left-to-right-Logistic-Regression-Nearest_fig3_327836487